MODULE P · OCR J277 2.5.1, 2.5.2 · CAMBRIDGE 0478 4.2.1-4.2.5

Languages and IDEs

High-level and low-level languages, **assembly language**, **compilers and interpreters** and **IDE tools**. **OCR J277: skip card 04 (P2)** and the IGCSE-only rows; assemblers are not required. OCR examines this on Paper 2, Cambridge on Paper 1. Spec versions as checked in November 2026.

01 High-level languages (P1) · both boards

What English-like keywords and statements (if, while, print). Close to how people write.

Translation Must be translated to machine code. 1 statement becomes many machine code instructions.

Example `total = price * quantity`: load, multiply, store. 1 statement, 3 instructions.

Used for Most programs: easier to write, read and debug; machine independent.

02 Low-level languages (P1) · both boards

What Close to the CPU's own instructions. Written for one type of processor.

Machine code Binary instructions the CPU executes directly, no translation.

Assembly Short codes (mnemonics) for those instructions. Also low-level.

Used for Device drivers, embedded systems: speed, small code, hardware control.

03 High-level vs low-level (P1) · both boards

Reading, writing High-level: easier to read, write and understand. Low-level: hard to read and write.

Debugging High-level: errors easier to find and fix. Low-level: errors harder to find and fix.

Machine independence High-level: can be translated for different processors (portable). Low-level: runs only on the processor type it was written for.

Hardware Low-level: direct manipulation of hardware and memory. High-level: no direct control.

Speed, memory Low-level can run faster and use less memory. High-level is **not** faster.

IGCSE only 0478 lists pros and cons under four headings: reading and writing, debugging, machine independence, direct manipulation of hardware.

04 Assembly language (P2) IGCSE only

What A form of low-level language that uses mnemonics in place of binary.

Mnemonic Short code for an instruction, e.g. LDA (load), ADD, STO (store).

Assembler Translates assembly language into machine code. Not a compiler or interpreter.

1 to 1 Generally 1 assembly instruction becomes 1 machine code instruction.

Example `LDA 20` (15), `ADD 21` (27), `STO 22`: stores 42. Made-up codes.

OCR J277 Assemblers not required.

05 Translators (P3) · both boards

Purpose Convert source code into machine code.

Why needed The CPU can only execute machine code; high-level code is not machine code.

Compiler Translates the whole program at once, before it runs. Produces an executable file.

Interpreter Translates and executes one line at a time, every time the program runs. No executable.

GCSE only J277 translators: compilers and interpreters only.

06 Compiler vs interpreter (P3) · both boards

Compiler

Errors: error report for the whole code; no executable until fixed.
Benefits: executable runs fast; users need no translator; source code not given away.
Drawbacks: compiling takes time; recompile after every change; executable runs on one processor type.

Interpreter

Errors: stops execution at the first error and reports it; carries on once it is fixed.
Benefits: error shown at the line it happens; test changes straight away.
Drawbacks: slower to run (translated every time); interpreter needed every run; source code must be shared.

07 Errors: one program (P3) · both boards

Program 10 lines; errors on lines 4 and 9.

Interpreter Lines 1-3 run. Stops at line 4 and reports it. Line 9 found only after line 4 is fixed.

Compiler All 10 lines translated first; nothing runs. One report lists lines 4 and 9.

500 runs Compiled: translated once. Interpreted: translated 500 times.

08 Which one when? (P3) · both boards

Developing Interpreter: errors at the line they occur; quick to test changes.

Final program Compiler: fast executable, no translator needed, source code protected.

IGCSE only 0478 names it: interpreter mostly used when developing; compiler translates the final program. Compiler gives an error report for the whole code; interpreter stops when an error is found.

09 IDE tools (P4) · both boards

IDE One program with the tools to write, translate, run, test and debug code.

Editor Enter and edit code. Line numbers, colour-coded keywords, auto-indent.

Error diagnostics Line number and message for each error. Breakpoints, stepping, watching variables.

Run-time env. Run the program inside the IDE to test it and see its output and run-time errors.

Translator Built-in compiler and/or interpreter: code to machine code.

10 IDE extras (P4)

Auto-completion IGCSE only Suggests and finishes keywords and identifiers as you type.

Auto-correction IGCSE only Automatically fixes common mistakes, e.g. a misspelt keyword.

Prettyprint IGCSE only Consistent indentation and colour-coding: easier to read, mistakes stand out.

GCSE only J277's four tools: editors, error diagnostics, run-time environment, translators. Practical use of them in at least one IDE.

01
Machine code and assembly language are both low-level. Low-level can be faster; high-level is easier.

02
Translators convert code into machine code, not "a language the computer understands".

03
Only a compiler makes an executable. Compiler: all errors in one report. Interpreter: stops at the first.

04
IDE answers: name the tool and say how it helps the programmer.